Middleware for Dynamic Adaptation of Component Applications

Boyana Norris,[1] Sanjukta Bhowmick,[1,2] Dinesh Kaushik, and[1]
Lois Curfman McInnes[1]

[1] Mathematics and Computer Science Division, Argonne National Laboratory,
9700 South Cass Ave., Argonne, IL 60439, U.S.A.
[2] Department of Applied Physics and Applied Mathematics, Columbia University,
200 S.W. Mudd Building, 500 W. 120th Street, New York, NY 10027, U.S.A.
[norris,bhowmick,kaushik,mcinnes]@mcs.anl.gov

Abstract. Component- and service-based software engineering approaches have been gaining popularity in high-performance scientific computing, facilitating the creation and management of large multidisciplinary, multideveloper applications, and providing opportunities for improved performance and numerical accuracy. These software engineering approaches enable the development of middleware infrastructure for computational quality of service (CQoS), which provides performance optimizations through dynamic algorithm selection and configuration in a mostly automated fashion. The factors that affect performance are closely tied to a component's parallel implementation, its management of parallel communication and memory, the algorithms executed, the algorithmic parameters employed, and other operational characteristics. We present the design of a component middleware CQoS architecture for automated composition and adaptation of high-performance component- or service-based applications. We describe its initial implementation and corresponding experimental results for parallel simulations involving time-dependent nonlinear partial differential equations.

1 Introduction

As computational science progresses toward ever more realistic multiphysics and multiscale applications, no single research group can effectively develop, select, or tune all of the components in a given application, and no single tool, solver, or solution strategy can seamlessly span the entire spectrum *efficiently*. Component- and service-based software development approaches help manage some of the complexity of developing such large scientific applications. Current component and service specifications, however, provide support only for basic manipulation of components and services, such as repositories, instantiation, connection, and execution. Common component interfaces and service specifications enable easy access to suites of independently developed algorithms and implementations, and dynamic composability facilitates switching among different implementations during runtime. The challenge then becomes how to

automatically make sound choices from among the available implementations and parameters, with suitable tradeoffs among performance, accuracy, mathematical consistency, and reliability. Such choices are important both for the initial composition and configuration of an application and for adaptive control during runtime.

With the increased availability of solution methods implemented as components or services, a major challenge is to ensure that the choice of one of many implementations of a particular interface produces a result of the desired quality within a reasonable amount of time. One can address this challenge by automating at least some of the process of selecting and configuring components, with the goal of minimizing execution time within a set of quality constraints. In order to provide such support, a specification is needed that describes the quality metrics, i.e., metadata for functional and nonfunctional properties and requirements of components. Furthermore, the performance of components must be monitored and recorded in a nonintrusive fashion. In addition, the performance data must be analyzed in order to construct performance models of individual components and whole applications, which can then be used by heuristics that take into account performance information and quality constraints in order to compose and adapt applications in an optimized fashion.

Computational Quality of Service. We are addressing this challenge by developing a high-level specification and corresponding middleware for *computational quality of service* (CQoS) [49], or the automatic selection and configuration of components to suit a particular computational purpose. CQoS extends the familiar concept of quality of service (QoS) in networking with domain-specific quality metrics and the ability to specify and manage characteristics of the application in a way that adapts to the changing computational environment. Traditional QoS emphasizes system-related performance effects such as CPU or network loads to implement application priority or bandwidth reservation in networking. Although performance is a shared general concern, high efficiency and parallel scalability are more significant requirements for high-performance scientific applications, along with algorithmic or problem-specific qualities, such as the level of solution accuracy achieved by a particular algorithm. This situation has motivated us to define an expanded notion of CQoS that better reflects the characteristics and needs of high-performance component- or service-based scientific applications.

Common Component Architecture. While our goal is a component-neutral or service-model-neutral CQoS architecture, our work to date on implementing CQoS middleware employs the Common Component Architecture (CCA) [4, 7, 17], which is designed specifically for the needs of parallel, scientific high-performance computing (an area where other component approaches are limited). A comprehensive description of the CCA, including a discussion of how it differs from other component models, is available [7]; here we present a brief overview of the CCA environment, focusing on the aspects most relevant to CQoS infrastructure.

The specification of the Common Component Architecture [16] defines the rights, responsibilities, and relationships among the various elements of the model. Briefly, the elements of the CCA model are as follows:

- *Components* are units of software functionality that can be composed together to form applications. Components encapsulate much of the complexity of the software inside a black box and expose only well-defined interfaces.
- *Ports* are the abstract interfaces through which components interact. Specifically, CCA ports provide procedural interfaces that can be thought of as a class or an interface in object-oriented languages, or a collection of subroutines, or a module in a language such as Fortran 90. Components may provide ports, meaning that they implement the functionality expressed in a port (called *provides* ports), or they may use ports, meaning that they make calls on a port provided by another component (called *uses* ports). Components that provide the same port(s) are considered functionally equivalent and can thus be used interchangeably.
- *Frameworks* manage CCA components as they are assembled into applications and executed. The framework is responsible for instantiating components, destroying instances, and connecting *uses* and *provides* ports without exposing the components' implementation details. The framework also provides a small set of standard services that are available to all components. The CCA implementation of the CQoS infrastructure described in this paper relies on the CCA specification and basic services to provide new middleware components for performance monitoring, analysis, and dynamic application adaptation.

Paper Organization. The remainder of this paper introduces our component middleware architecture for CQoS. Section 2 discusses related work, and Section 3 introduces several high-performance scientific applications that motivate this research, with emphasis on simulations involving the parallel solution of time-dependent, nonlinear partial differential equations (PDEs). Section 4 describes our approach and implementation, and Section 5 presents preliminary experimental results. Section 6 discusses conclusions and directions of future work.

2 Related Work

Adaptive software for scientific computing is an area of emerging research, as evidenced by numerous recent projects and related work [14, 18, 21–26, 31, 35, 36, 39, 40, 43, 53, 55, 57, 60, 62–66, 69]. Many approaches to addressing different aspects of adaptive execution are represented in these projects, from compiler-based techniques to development of new numerical adaptive algorithms.

Three approaches of interest for specifying semantic information are models, contracts, and service-level agreements. Furmento et al. [28] as well as Gu

and Nahrstedt [30] discuss performance models and their use in overall component application assembly at runtime within the context of distributed environments; Beugnard et al. [8] define a general model of software contracts and discuss approaches for making components contract-aware. Similarly, the SAMcode model of adaptable mobile agents [1] allows the specification of contracts — consisting of one precondition and one postcondition — for each adaptable method. Violations are used to select from different implementations of a method at runtime. The GlueQoS work of Wohlstadter et al. [68] focuses on mediating quality-of-service requirements — specified as assertions — between clients and Web services. Bennett et al. [6] discuss the need for service-level agreements for defining the terms and conditions of use, with agreements providing a minimum of coupling between components. They also emphasize the importance of characterizing relevant component features to ensure both the correct use and provision of services. Raje et al. [50] describe a QoS framework for distributed, heterogeneous components and provide a catalog of QoS metrics [13]. The Software-Implemented Fault Tolerance (SIFT) environment for Adaptive Reconfigurable Mobile Objects of Recovery (ARMOR) processes [67] relies on their model for functional reconfiguration to adjust application behavior to meet dependability requirements. In this case adaptation is accomplished through user-specified assertion checks at critical execution points and the use of microcheckpointing to adjust application state accordingly.

In the area of dynamic adaptation based on monitoring application behavior, Reiner and Pinkerton [52] explore dynamically changing control parameters to improve operating system performance and use experiments to determine improved settings. They develop a methodology for adaptive tuning as well as algorithm, policy, and (fixed) parameter selection. Whisnant et al. [67] rely on human intervention to deal with reconfiguration after a problem is detected at runtime. Feather et al. [27], however, use event monitoring of behavioral deviations and changing environmental conditions to reconcile the intended system behavior with individual requirements at runtime. In these cases, monitoring an application at runtime involves checking control parameters and monitoring events, including application failure.

Unlike these efforts, our approach is specifically targeted at large-scale parallel computations and relies on high-level interface specifications and technologies tailored for scientific computing. In designing our CQoS interfaces and middleware components, we rely on the existing high-performance infrastructure provided by the CCA, in which multiple component implementations conforming to the same external interface standard are interoperable and the runtime system ensures that the overhead of component substitution is negligible.

3 Motivating Applications and Algorithms

As discussed in [45], a variety of high-performance scientific applications motivate the development of infrastructure for computational quality of service,

including mesh partitioning in combustion simulations [58,59], resource management in quantum chemistry [38], and the solution of linear systems that arise in nonlinear PDE-based simulations in domains such as high-energy accelerators, computational fluid dynamics, and radiation transport. A common feature of these large-scale, long-running simulations is the combination of diverse numerical capabilities, such as physics models, discretizations, linear solvers, nonlinear solvers, and optimization solvers, each having multiple implementations with varying degrees of fidelity, robustness, efficiency, and scalability. Moreover, it is not generally known a priori which combination of implementations will be best suited for a particular problem instance and computational environment.

Before explaining in Section 4 our approach to handling these issues with CQoS middleware, we briefly introduce two parallel PDE-based applications in which a significant fraction of overall execution time is devoted to the solution of large-scale, sparse linear systems. In this context, CQoS focuses on selecting and configuring linear solvers (typically preconditioners and Krylov methods) based on the context of the overall simulation. Because the properties of linear systems in time-dependent or nonlinear applications may significantly change during the course of a simulation, CQoS-enabled adaptive multimethod solvers have promise to improve robustness and reduce overall time to solution [10–12, 44]. Section 5 presents experimental results of CQoS-enabled adaptive linear solvers for these two applications.

Transonic Euler Flow. We consider the solution of the unsteady compressible three-dimensional Euler equations using PETSc-FUN3D [3], an unstructured mesh code originally developed by W. K. Anderson [2] and subsequently parallelized using MeTiS [34] for mesh partitioning and the PETSc library [5] for the preconditioned Newton-Krylov family of implicit solution schemes. This code uses a finite volume discretization with a variable-order Roe scheme on a tetrahedral, vertex-centered mesh; details of the discretization and parallelization are discussed in [3]. We explore the standard aerodynamics test case of transonic flow over an ONERA M6 wing using the frequently studied parameter combination of a freestream Mach number of 0.839 with an angle of attack of 3.06^o. The robustness of solution strategies is particularly important for this model because of the so-called λ-shock that develops on the upper wing surface, as depicted in Figure 1. The PDEs are initially discretized by using a first-order scheme; but once the shock position has settled down, a second-order discretization is applied.

Radiation Transport. Under the assumptions of isotropic radiation with no frequency dependence, transport through a material characterized by spatially varying atomic number (Z) and thermal conductivity (κ) can be modeled by the following coupled nonlinear equations in radiation energy density (E) and material temperature (T):

$$\frac{\partial E}{\partial t} - \nabla \cdot (D_E \nabla E) = \sigma_a (T^4 - E), \quad \frac{\partial T}{\partial t} - \nabla \cdot (D_T \nabla T) = -\sigma_a (T^4 - E) \quad (1)$$

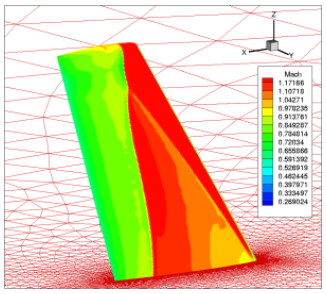

Fig. 1. Mach contours on the ONERA M6 wing at freestream Mach number = 0.839.

with

$$\sigma_a = \frac{Z^3}{T^3}, \quad D_E(E,T) = \frac{1}{3\sigma_a + \frac{|\nabla E|}{|E|}}, \quad \text{and } D_T(T) = \kappa T^{\frac{5}{2}}. \tag{2}$$

In order to restrict the maximum speed of propagation to the speed of light, the above formula for diffusivity D_E includes Wilson's flux limiter $|\nabla E|/|E|$ [29, 46], which makes these governing equations highly nonlinear. The spatial discretization in [29] employs Galerkin finite elements with linear piecewise continuous basis functions over simplices in 2D and 3D. Temporal integration is done by a solution-adaptive implicit Euler method. This code shows excellent scalability on the TeraGrid, Blue-Gene, and System X platforms [29].

The cross section of the computational domain in 3D is the unit square, with a radiation flux incident on the left boundary. The atomic number is location dependent (only in x and y):

$$Z(x,y,z) = \begin{cases} 10 \text{ for } \frac{1}{3} \le x \le \frac{2}{3} \text{ and } \frac{1}{3} \le y \le \frac{2}{3}, \\ 1 \quad \text{elsewhere.} \end{cases} \tag{3}$$

The boundary conditions for Equations (1) are set by imposing a constant radiation field at $x = 0$:

$$\mathbf{n} \cdot D_E \nabla E + \frac{E}{2} = 2 \text{ at } x = 0 \text{ and } \mathbf{n} \cdot D_E \nabla E + \frac{E}{2} = 0 \text{ at } x = 1,$$
$$\text{and } \mathbf{n} \cdot \nabla E = 0 \text{ at } y = 0 \text{ and } y = 1,$$

where $\mathbf{n}$ is the outward unit normal to the boundary, as in [42]. The temperature contours showing the propagation of the thermal front at $t = 1$ and $t = 3$ are given in Figure 2.

Algorithmic Overview. Both of these nonlinear PDE-based applications employ Newton-Krylov methods (see, e.g., [47]) within the PETSc library [5] to solve nonlinear equations of the form $f(u) = 0$, where $f : R^n \to R^n$, at each timestep of the simulation. We use a two-step sequence of (approximately) solving the Newton correction equation

$$(f'(u^{\ell-1}))\ \delta u^{\ell} = -f(u^{\ell-1}) \tag{4}$$

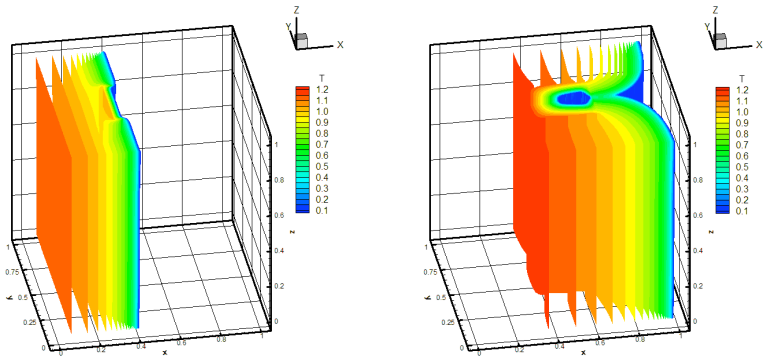

Fig. 2. Evolution of material temperature in time for a 3D example with a tetrahedral mesh of 237,160 vertices and 1,264,086 elements. The left figure shows the temperature contours at $t = 1$, while the right shows temperature at $t = 3$.

and then updating the iterate via $u^\ell = u^{\ell-1} + \delta u^\ell$. If the Jacobian matrix f' is poorly conditioned, the Krylov method will require an unacceptably large number of iterations. The system (4) can be transformed into the equivalent form $B^{-1}f'(u^{\ell-1})\delta u^\ell = -B^{-1}f(u^{\ell-1})$ through the action of a preconditioner, B, whose inverse action approximates that of f', but at smaller cost. We thus consider in Section 5 a variety of different preconditioners and Krylov methods, with a goal of achieving low computational cost and scalable parallelism.

The radiation transport code uses an analytical second-order accurate Jacobian matrix f', where the preconditioner is derived from the same matrix. In contrast, the compressible Euler application employs matrix-free Newton-Krylov methods (see, e.g., [15]), with which we compute the action of the Jacobian on a vector v by directional differencing of the form $f'(u)v \approx \frac{f(u+hv)-f(u)}{h}$, where h is a differencing parameter. We use a first-order analytic discretization to compute the corresponding preconditioning matrix.

For both applications, the time to solve the Newton correction equation (4), is a significant fraction of overall execution time (about 35% for the radiation transport code and about 75% for the compressible Euler code). Moreover, as further discussed in Section 5, changes in the numerical characteristics of the linear systems reflect the changing nature of the simulations. For example, the use of pseudo-transient continuation [37] in the compressible Euler application generates linear systems that become progressively more difficult to solve as the simulation advances (see Figure 5). Likewise, the linear and nonlinear systems become progressively more challenging as the timesteps (based on dynamical scales of the problem) increase in the radiation transport application (see Figure 6). Consequently, both applications provide strong motivation for the development of CQoS middleware to support multimethod adaptive linear solver algorithms.

4 Computational Quality of Service for Components

This section describes in detail our approach to defining and implementing computational quality of service for components, which was introduced in Section 1.

4.1 Approach

We begin by reviewing the main requirements for enabling computational quality of service support in component-based scientific applications. First, we must be able to monitor the performance of individual components without requiring manual code modifications. The performance data must be collected and stored for later access. Performance information alone, however, is not sufficient to enable effective adaptive strategies in numerical software. Thus, we must also identify nonfunctional *quality* metrics, which are problem- or algorithm-specific, and nonintrusively record the resulting metadata corresponding to these metrics along with the performance data. The accumulated runtime information can also be augmented with a priori or source-based analysis of algorithms, whenever such are available. Given this combined database, the application performance can be characterized by means of different approaches, including machine learning and statistics. The results of such analyses would be used to construct performance models for individual components or whole applications. Finally, there must be a mechanism for specifying dynamic adaptation (component reconfiguration or substitution) based on these performance models and additional problem metadata.

Our goal is to address these requirements by providing middleware for component- or service-oriented frameworks, with the CCA as our initial target component model. We rely on the discipline of interface definition, which is at the core of both component- and service-based software engineering approaches, in order to automate the gathering of performance and other data, as well as to enable automated dynamic reconfiguration and substitution of computational units, expressed as either components or services.

The principal purpose of CQoS in the context of high-performance computing is to provide methodology and support for optimizing the time to solution of component- or service-based applications. We have identified two main ways through which this goal can be achieved: (1) by optimizing the selection component instances for the initial composition of an application and (2) by dynamically reconfiguring or substituting component instances.

4.2 Architecture

To support CQoS in scientific applications, we describe a high-level architecture that is not dependent on a particular component or service model (Figure 3). This architecture consists of two main parts: (1) *measurement and analysis* components, which are responsible for monitoring and gathering performance information and other metadata and for operating on and augmenting these

data, and (2) *control infrastructure*, which consists of components that implement domain-specific adaptive strategies, along with runtime services for component reconfiguration and substitution. Work that has contributed to the design of this architecture is described in [32, 41, 45, 49, 51, 61].

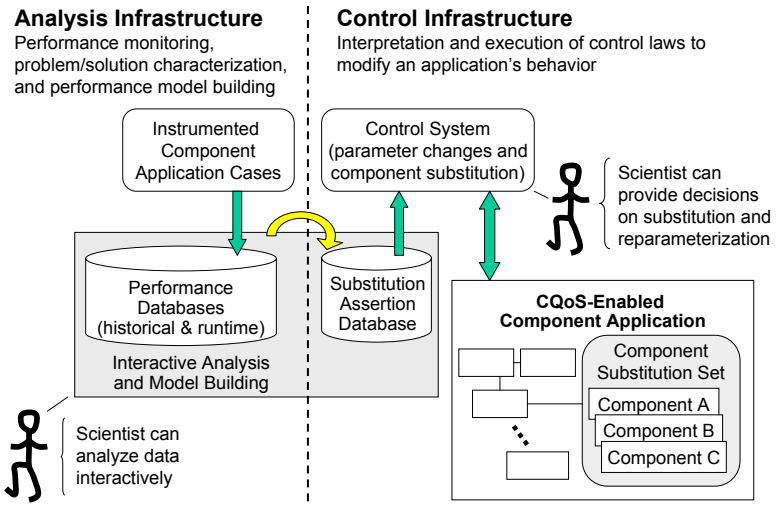

Fig. 3. CQoS middleware architecture overview.

The monitoring portion of the infrastructure deals with collecting performance data, as well as domain-specific metadata that is related to or may impact the performance of an application. The two main requirements for the monitoring and data gathering support are that (1) minimal or no code changes are needed to enable monitoring, and (2) the overhead of the data gathering functionality is negligible with respect to the rest of the computation.

The analysis infrastructure consists of components that operate on any available performance data and associated metadata for individual components or whole applications. Different types of analyses, for example statistical or machine learning, can be incorporated in order to derive a characterization, or model, of the performance of an application or its constituent components. The models generated by analyses or provided by developers are stored in the persistent performance database, along with references to and from the performance data from which they were generated. When performance models of individual components are available, analysis components for generating whole-application models, such as those described in [41], can be employed to derive a performance model for an application composed from these components. Another

source of performance models is analytic closed expressions for the execution time provided by users or source code analysis tools. Such models are usually less accurate and are limited by the complexity of the parameterization used. While the accuracy of performance models can vary greatly, the availability of such models is crucial for enabling CQoS support, both for initial application composition and for runtime adaptation.

4.3 Implementation

While the high-level architecture described in Section 4.2 represents our vision of the general structure of middleware for CQoS support, it does not dictate low-level implementation details; thus, specialized implementations can be provided for different computational environments. Our current focus is on tightly coupled high-performance architectures because the majority of our motivating applications are written for such platforms using a single-program multiple-data (SPMD) programming model. This does not preclude implementations targeting more loosely coupled Grid-based environments, which would be able to reuse at least some of the middleware analysis and control infrastructure implementations.

We have implemented portions of the architecture described in Section 4.2; an early prototype is described in [48]. The initial implementation provides automated performance instrumentation of C++ CCA components using the Tuning and Analysis Utilities (TAU) software [54]. In addition to providing portable instrumentation capabilities, TAU provides a database format definition, the Performance Data Management Framework (PerfDMF) [33], for storing performance data and other application metadata. In our initial implementation, we leveraged the performance monitoring approach described in [41], extending it to collect component-specific metadata in addition to performance metrics. For example, in an application involving the solution of a nonlinear PDE using a Newton-based solver, we monitor and record the number of nonlinear iterations. Furthermore, we implemented context-sensitive monitoring of performance and related metadata; for example, within each nonlinear solution, we monitor and record the linear solver algorithm used, the preconditioner type, and the number of linear iterations (for iterative Krylov subspace solvers). In the database, performance and algorithm-specific execution metadata is associated with an application *experiment*, which is defined as an application instance consisting of a set of component instances and their configurations. Including component configuration parameters in the CQoS metadata is crucial because they can significantly change the performance characteristics of an application; different parameter values can result in drastically different performance for the same set of components. For example, in a driven cavity fluid dynamics simulation [19], the lid velocity and Grashof number determine to a large degree the difficulty of the problem instance; in addition, algorithmic parameters, such as the initial CFL number, affect the convergence speed and thus total execution time.

Our initial use case for the CQoS infrastructure focuses on enabling adaptive algorithms for the solution of nonlinear PDEs. In particular, we target multi-method approaches to adaptive linear system solution, such as those described in [10, 11, 44]. While it is possible to define application-independent adaptive strategies based only on general data mining of the historical performance information, our initial approach is based on developing application or domain-specific analysis and corresponding control components, which employ both the performance information and the associated application-specific metadata. The main disadvantage of this approach is that it is not generally applicable in a black-box fashion to arbitrary applications for which we have gathered sufficient performance data. A significant advantage, however, is that by focusing on developing analysis algorithms and adaptive strategies for a particular application domain, we are able to construct much more accurate performance models and more detailed control components, resulting in greater potential performance improvements.

An implementation of a subset of the CQoS infrastructure for adaptive linear solver components in time-dependent nonlinear PDE-based applications is illustrated in Fig. 4. New middleware components for monitoring the performance of nonlinear and linear solver components and for recording algorithm-specific metadata are introduced. An adaptive strategy component serves as a proxy for a linear solver component, presenting the same public interface as a non-adaptive linear solver component. The adaptive strategy can be implemented as one or more components; in this case, it combines the analysis and control portions of the CQoS architecture for selecting among different linear solver algorithms throughout the nonlinear solution process.

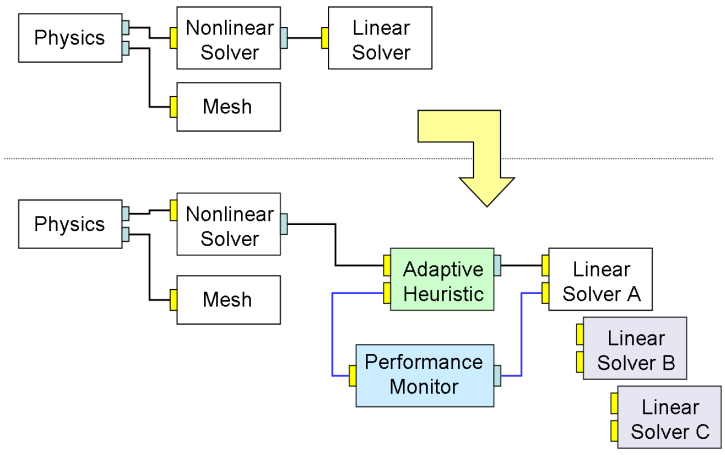

Fig. 4. Adaptive linear solver components in nonlinear PDE applications: a typical component application without adaptivity (top) and the same application with support for performance monitoring and adaptive linear solvers (bottom).

We have developed a number of adaptive heuristics with various degrees of generality. For example, an approach that is generally applicable to Newton-based nonlinear system solution is to monitor the nonlinear rate of convergence and switch linear solvers when a given threshold is reached. Approaches that exploit some domain or application-specific knowledge in addition to the algorithmic metadata can result in more effective adaptive behavior. For example, a change in a physical parameter that is known to affect the characteristics of the linear systems can be used to trigger linear solver component substitution. In general, when designing new adaptive strategies, we exploit both application-specific and algorithmic parameters whenever possible. Initial heuristics for a new application domain may be fully manual, using human insight to guide the adaptation, and gradually evolving into more automated strategies that include more sophisticated analysis components.

5 Experimental Results

We used the Jazz cluster at Argonne National Laboratory to run the simulations for the compressible Euler and radiation transport applications introduced in Section 3. The cluster has a Myrinet 2000 network and 2.4 GHz Pentium Xeon processors with 1-2 GB of RAM. We experimented with one problem instance from each motivating application, both of which required the solution of large-scale linear systems with sparse coefficient matrices. The compressible Euler code generated Jacobian matrices of rank approximately 1.8×10^6 with 1.3×10^8 nonzeros, while the radiation transport code generated matrices of rank 4.5×10^5 with 6.3×10^6 nonzeros. We ran the simulations on four processors using *base solvers* composed of various Krylov methods and subdomain solvers for a block Jacobi preconditioner with one block per processor.

We compare the performance of the simulations using adaptive solvers with that of the base solvers. Use of adaptive solvers can improve the overall performance by dynamically selecting the most appropriate method to match the needs of the current linear system, such as combining more robust (but more costly) methods when needed in particularly challenging phases of solution with faster (though less powerful) methods in other phases. Adaptive solvers can be defined by the heuristic employed for method selection. The efficiency of an adaptive heuristic depends on how appropriately it determines *switching points*, or the iterations at which to change linear solvers. In this paper we employed sequence-based adaptive heuristics, which rely on a predetermined sequence of linear solvers and then "switch up" to a more robust but more costly method or "switch down" to a cheaper but less powerful method as needed during the simulation. The sequence of base solvers is ordered by the *average time per nonlinear iteration* required by each solver. This measurement provides a rough estimate of the strength of the linear solver.

5.1 Transonic Euler Flow

Solver Specifications. We employed the following four base solvers, consisting of a Krylov method and block Jacobi preconditioner with one block per processor with the specified subdomain solver:

1. GMRES with SOR as a subdomain solver, designated as GMRES-SOR
2. Bi-conjugate gradient squared (BCGS) with no-fill incomplete factorization (ILU(0)) as the subdomain solver, called BCGS-ILU0
3. Flexible GMRES (FGMRES) with ILU(0) as the subdomain solver, designated as FGMRES-ILU0
4. GMRES with ILU(1) as a subdomain solver, designated as GMRES-ILU1

Adaptive Heuristics. The compressible Euler code uses pseudo-transient continuation [37] to advance the solution to an assumed steady state. The CFL number [37] provides a good indication of the relative difficulty of the resulting Newton system, with lower CFL numbers indicating systems that are better conditioned and thus easier to solve than those with higher CFL numbers. The left-hand graph of Figure 5 shows that the change in CFL number is inversely reflected by the change in the nonlinear residual norm. Thus, the nonlinear residual norm is a good indicator of the level of difficulty of solving its corresponding Newton correction equation: the lower the residual norm, the more difficult the linear system. Based on trial runs of the application, we divided the simulation into four sections: (a) $||f(u)|| \geq 10^{-2}$, (b) $10^{-4} \leq ||f(u)|| < 10^{-2}$, (c) $10^{-10} \leq ||f(u)|| < 10^{-4}$, and (d) $||f(u)|| < 10^{-10}$. Whenever the simulation crosses from one section to another, the adaptive method switches up or down accordingly.

The relative linear convergence tolerance was 10^{-3}, and the maximum number of iterations for any linear solve was 30. We ordered these methods for use in the adaptive solver as 1, 2, 3, 4, according to the average time taken per nonlinear iteration in the first-order discretization phase of the simulation, which can serve as a rough estimate of the strength of the various linear solvers for this application.

Results. The right-hand graph of Figure 5 show the switching points among these methods in the adaptive polyalgorithmic approach. The simulation starts with method 1, then switches to method 2 at the next iteration. The switch to method 3 occurs at iteration 25. The discretization then shifts to second order at iteration 28, and the initial linear systems become easier to solve. The adaptive method therefore switches down to method 2. From this point onward, the linear systems become progressively more difficult to solve as the CFL number increases; the adaptive method switches up to method 3 in iteration 66 and method 4 in iteration 79. The last change is accompanied by an increase in the time taken for the succeeding nonlinear iteration. This increased time is devoted to setting up the new preconditioner, which in this case changes the block Jacobi subdomain solver from ILU(0) to ILU(1) and consequently requires more time

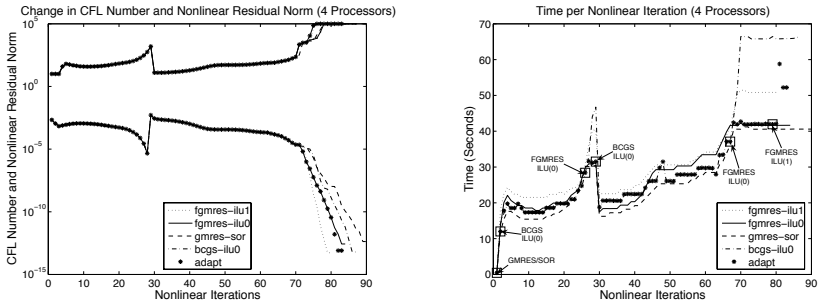

Fig. 5. *Left:* Convergence rate (lower plot) and CFL number (upper plot) for the base and adaptive solvers on 4 processors. *Right:* Time per nonlinear iteration for the base and adaptive solvers on 4 processors.The labeled square markers indicate when linear solvers changed in the adaptive algorithm.

for the factorization phase. The execution time of the adaptive polyalgorithmic scheme is 3% better than the fastest base method (FGMRES-ILU0) and 20% better than the slowest one (BCGS-ILU0).

5.2 Radiation Transport

Solver Specifications. We employed the following four base solvers, consisting of a Krylov method and block Jacobi preconditioner with one block per processor with the specified subdomain solver:

1. GMRES with SOR as a subdomain solver, designated as GMRES-SOR
2. Flexible GMRES (FGMRES) with ILU(0) as a subdomain solver, designated as FGMRES-ILU0
3. GMRES with ILU(0) as a subdomain solver, designated as GMRES-ILU0
4. Bi-conjugate gradient squared (BCGS) with with ILU(0) as a subdomain solver, designated as BCGS-ILU0

The relative linear convergence tolerance was 10^{-3} and the maximum number of iterations for any linear solve was 80.

Adaptive Heuristics. In contrast to the previous application, the radiation transport code completely solves a nonlinear system at *each* time step. The number of nonlinear iterations (4-10) required for convergence of each nonlinear system is quite small, rendering the use of adaptive solvers specific to each nonlinear solution unnecessary. However, the difficulty of the nonlinear systems themselves varies over the timesteps, and this factor can be utilized to generate adaptive solvers. Thus, the linear solvers stay constant during the solution of each nonlinear system but may change as the nonlinear equations change with the timesteps.

The left-hand graph of Figure 6 plots the timestep size with respect to the simulation's progress. We sampled the average time per iteration at timestep

intervals of 1 second. The right-hand graph shows the average time per iteration of the base solvers at these intervals. We note that although solvers 1 and 4 remain in their highest and lowest positions, the relative order of solvers 2 and 3 varies, especially between the time steps 2 and 3. Thus the sequence of solvers is 1, 2, 3 and 4, except in the interval (2-3 seconds), where the sequence is 1, 3, 2 and 4.

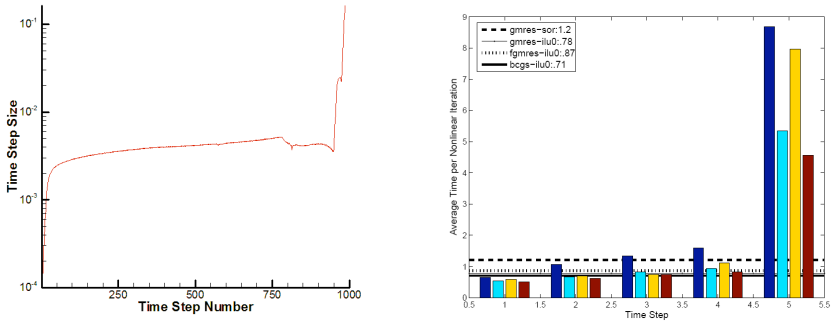

Fig. 6. *Left:* Change in timestep size over the simulation. *Right:* Change of average time per nonlinear iteration as simulation progresses. From left to right the bar indicates average time per iteration of solvers 1, 2, 3, and 4 within the timestep interval given in the x-axis. The legend shows the average time for each solver, over the entire simulation.

In addition to the timestep, a good indicator of the difficulty of the problem is the ratio of the linear iterations to nonlinear iterations. As the relative difficulty of the nonlinear system increases, this ratio increases correspondingly.

Results. We experimented with an automated adaptive solver, where the linear solver changes with change in the ratio of the linear to nonlinear iterations increases by 10%. Generally we switched to a faster solver if the increase was more than 10%; however, the ratio increased significantly in the first few step up to 30%. Therefore we switched from solver 1 to solver 3 skipping the intermediate solver 2. Another aberration to this rule was in the (2-3 second) interval where, since solvers 2 and 4 have nearly the same average time per iteration, we switched from solver 4 to solver 2 when the ratio increased by 10%. Since a switch in solvers can potentially increase the time, chiefly because of data structure manipulations needed when resetting the Krylov method and preconditioner, we kept the solver fixed for a window of at least four timesteps and then switched if necessary.

The solver switches shown in Figure 7 are as follows. The simulation begins with method 1 and switches to method 3 at time step 6, and then to method

4 at timestep 10. The next switch occurs at time step 552 to method 2. Then at timestep 740 the solver is changed to method 3, and finally at step 744 the solver becomes method 4 and this is maintained to the end. The automated adaptive solver is 1.2% better than the fastest base method (BCGS-ILU(0)) and 42.0% better than the slowest method (GMRES-SOR).

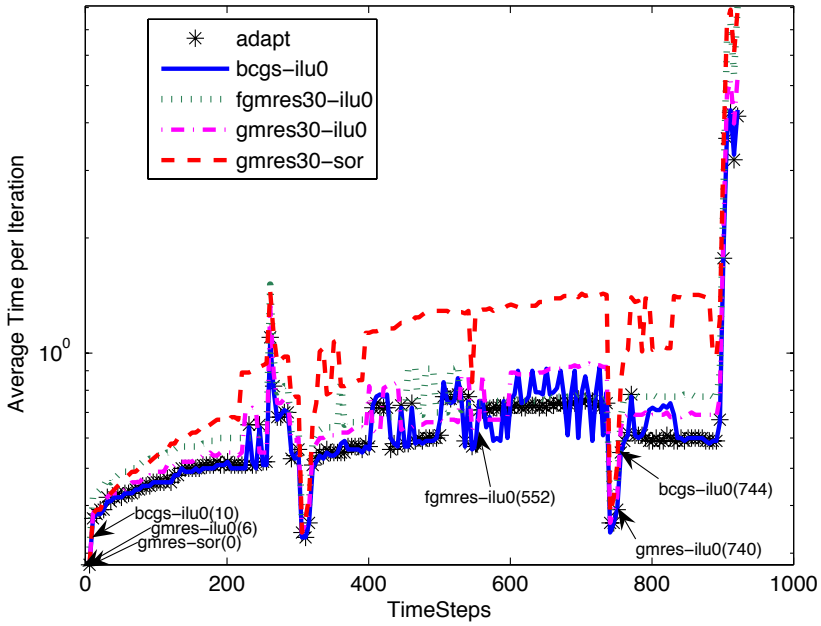

Fig. 7. Time per nonlinear iteration for the base and adaptive solvers on 4 processors. The markers indicate when linear solvers changed in the adaptive algorithm.

These preliminary experiments with the two motivating applications highlight the promise of adaptive solvers in the context of long-running simulations in which a single algorithm may not perform best throughout the entire simulation. These experiments also emphasize that solver performance differs considerably across application domains. For example, while BCGS-ILU(0) performed well for the radiation transport code, it was not the best performer for the transonic Euler code. The experiments also show that heuristics for adaptive multimethod solvers depend on the nature of the application.

Ongoing work includes applying these insights in adaptive strategies to larger problem instances of the radiation transport and transonic Euler applications. We are also working to incorporate scalable solver components [56] under development by the Terascale Optimal PDE Simulations (TOPS) project [20], which define a common interface through which one can provide easy access

to a broad range of scalable solvers developed by different groups at different institutions.

6 Conclusions

Motivated by the emerging needs of high-performance component-based scientific applications, we have introduced a general middleware architecture for computational quality of service, which provides support for runtime adaptation of component- or service-based applications with the goal of reducing the overall time to solution. We described an initial implementation of the CQoS architecture for CCA components, which provides support for adaptive algorithms, such as linear system solution. We demonstrated the effectiveness of this adaptive approach on parallel simulations of radiation transport and transonic Euler flow. While our current emphasis is on SPMD component applications, the overall architecture can be implemented in other contexts, such as distributed components and service-based applications.

Our current work focuses on refining the initial implementation to conform more closely to the CQoS overall architecture, including separation of analysis and control middleware components, as well as a robust implementation of the database management components. Future plans include adding more general analysis algorithms for extracting performance characteristics using statistical and machine learning methods [9] and leveraging related work by Eijkhout and Fuentes [26] on matrix characterization and metadata. We also will continue to explore the use of our CQoS approach in new application domains.

Acknowledgments

This work was supported by the U.S. Department of Energy under Contract W-31-109-Eng-38 and by the National Science Foundation award 04-06403. We thank the members of the Common Component Architecture Forum for many stimulating discussions about high-performance software components. We also thank Barry Smith for making some enhancements to the PETSc library to facilitate the development of adaptive solvers.

References

1. Noriki Amano and Takuo Watanabe. A software model for flexible and safe adaptation of mobile code programs. In *Proceedings of the International Workshop on Principles of Software Evolution*, pages 57–61, Orlando, FL, May 2002.
2. W. K. Anderson and D. Bonhaus. An implicit upwind algorithm for computing turbulent flows on unstructured grids. *Computers and Fluids*, 23(1):1–21, 1994.

3. W. K. Anderson, W. D. Gropp, D. K. Kaushik D. E. Keyes, and B. F. Smith. Achieving high sustained performance in an unstructured mesh CFD application. In *Proceedings of Supercomputing 1999*. IEEE Computer Society, 1999. Gordon Bell Prize Award Paper in Special Category.
4. R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski. Toward a Common Component Architecture for high-performance scientific computing. In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing*, 1999.
5. S. Balay, K. Buschelman, W. Gropp, D. Kaushik, M. Knepley, L. McInnes, Barry F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 2.3.2, Argonne National Laboratory, 2006. `http://www.mcs.anl.gov/petsc`.
6. K. Bennett, P. Layzell, D. Budgen, P. Brereton, L. Macaulay, and M. Munro. Service-based software: The future for flexible software. In *Proceedings of the 7th Asia-Pacific Software Engineering Conference (APSEC 2000)*, pages 214–221, 2000.
7. D. E. Bernholdt, B. A. Allan, R. Armstrong, F. Bertrand, K. Chiu, T. L. Dahlgren, K. Damevski, W. R. Elwasif, T. G. W. Epperly, M. Govindaraju, D. S. Katz, J. A. Kohl, M. Krishnan, G. Kumfert, J. W. Larson, S. Lefantzi, M. J. Lewis, A. D. Malony, L. C. McInnes, J. Nieplocha, B. Norris, S. G. Parker, J. Ray, S. Shende, T. L. Windus, and S. Zhou. A component architecture for high-performance scientific computing. Intl. J. High-Perf. Computing Appl., in press, 2006.
8. Antoine Beugnard, Jean-Marc Jézéquel, Noël Plouzeau, and Damien Watkins. Making components contract aware. *IEEE Computer*, 32(7):38–45, July 1999.
9. S. Bhowmick, V. Eijkhout, Y. Freund, E. Fuentes, and D. Keyes. Application of machine learning to selecting solvers for sparse linear systems. submitted to International Journal of High Performance Computing Applications.
10. S. Bhowmick, D. Kaushik, L. McInnes, B. Norris, and P. Raghavan. Parallel adaptive solvers in compressible PETSc-FUN3D simulations. In *Proceedings of the 17th International Conference on Parallel CFD*, 2005. Also available as Argonne National Laboratory preprint ANL/MCS-P1283-0805.
11. S. Bhowmick, L. C. McInnes, B. Norris, and P. Raghavan. The role of multi-method linear solvers in PDE-based simulations. In *Lecture Notes in Computer Science*, volume 2667, pages 828–839, 2003. Computational Science and Its Applications-ICCSA 2003.
12. S. Bhowmick, P. Raghavan, L. C. McInnes, and B. Norris. *Faster PDE-Based Simulations Using Robust Composite Linear Solvers*, volume 20, pages 373–387. 2004.
13. G. J. Brahnmath, R. R. Raje, A. M. Olson, M. Auguston, B. R. Bryant, and C. C. Burt. A quality of service catalog for software components. In *Proceedings of the Southeastern Software Engineering Conference. http://www.ndiatvc.org/SESEC2002/*, 2002.
14. R. Bramley, D. Gannon, T. Stuckey, J. Villacis, J. Balasubramanian, E. Akman, F. Berg, S. Diwan, and M. Govindaraju. The Linear System Analyzer. In *Enabling Technologies for Computational Science*. Kluwer, 2000.
15. P. N. Brown and Y. Saad. Hybrid Krylov methods for nonlinear systems of equations. *SIAM Journal on Scientific and Statistical Computing*, 11:450–481, 1990.
16. CCA Forum. CCA specification. `http://cca-forum.org/specification/`, 2006.

17. CCA Forum homepage. `http://www.cca-forum.org/`, 2006.
18. R. Chowdhary, P. Bhandarkar, and M. Parashar. Adaptive QoS management for collaboration in heterogeneous environments. In *Proceedings of the 16th International Parallel and Distributed Computing Symposium (IEEE, ACM), 11th Heterogeneous Computing Workshop*, Fort Lauderdale, FL, 2002.
19. T.S. Coffey, C.T. Kelley, and D.E. Keyes. Pseudo-transient continuation and differential algebraic equations. *SIAM J. Sci. Comp*, 25:553–569, 2003.
20. D. Keyes (PI). Towards Optimal Petascale Simulations (TOPS) Center. `http://tops-scidac.org/`, 2006.
21. J. D. de St. Germain, John McCorquodale, Steven G. Parker, and Christopher R. Johnson. Uintah: A massively parallel problem solving environment. In *Proceedings of the Ninth IEEE International Symposium on High Performance and Distributed Computing*, August 2000.
22. J. D. de St. Germain, A. Morris, S. G. Parker, A. D. Malony, and S. Shende. Integrating performance analysis in the Uintah software development cycle. In *Fourth International Symposium on High Performance Computing (ISHPC-IV)*, pages 190–206, May 15-17 2002.
23. Jack Dongarra and Victor Eijkhout. Self-adapting numerical software and automatic tuning of heuristics. In *Proceedings of the International Conference on Computational Science*, 2003.
24. Jack Dongarra and Victor Eijkhout. Self-adapting numerical software for next generation applications. *International Journal of High Performance Computing Applications*, 17:125–131, 2003. also LAPACK Working Note 157, ICL-UT-02-07.
25. Thomas Eidson, Jack Dongarra, and Victor Eijkhout. Applying aspect-orient programming concepts to a component-based programming model. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS) April 22–26, 2003, Nice, France*, 2003.
26. V. Eijkhout and E. Fuentes. A proposed standard for matrix metadata. Technical Report ICL-UT 03-02, University of Tennessee, 2003.
27. M. S. Feather, S. Fickas, A. van Lamsweerde, and C. Ponsard. Reconciling system requirements and runtime behavior. In *Proceedings of the 9th International Workshop on Software Specification and Design*, pages 50–59, April 1998.
28. N. Furmento, A. Mayer, S. McGough, S. Newhouse, T. Field, and J. Darlington. Optimisation of component-based applications within a Grid environment. In *Proceedings of SC2001*, 2001.
29. W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith. A parallel implicit solver for diffusion limited radiation transport equations, 2006. Accepted for the Proceedings of the 16th International Conference on Domain Decomposition Methods.
30. X. Gu and K. Nahrstedt. A scalable QoS-aware service aggregation model for peer-to-peer computing Grids. In *Proceedings of HPDC 2002*, 2002.
31. E. N. Houstis, A. C. Catlin, J. R. Rice, V. S. Verykios, N. Ramakrishnan, and C. E. Houstis. A knowledge/database system for managing performance data and recommending scientific software. *ACM Transactions on Mathematical Software*, 26(2):227–253, 2000.
32. P. Hovland, K. Keahey, L. C. McInnes, B. Norris, L. F. Diachin, and P. Raghavan. A quality of service approach for high-performance numerical components. In *Proceedings of Workshop on QoS in Component-Based Software Engineering, Software Technologies Conference*, Toulouse, France, 2003. Also avail-

able as Argonne National Laboratory preprint ANL/MCS-P1028-0203 via `ftp://info.mcs.anl.gov/pub/tech_reports/reports/P1028.pdf`.
33. K. A. Huck, A. D. Malony, R. Bell, and A. Morris. Design and implementation of a parallel performance data management framework. In *Proc. International Conference on Parallel Processing (ICPP 2005)*. IEEE Computer Society, 2005.
34. G. Karypis and V. Kumar. A fast and high quality scheme for partitioning irregular graphs. *SIAM Journal of Scientific Computing*, 20:359–392, 1999.
35. K. Keahey, P. Beckman, and J. Ahrens. Ligature: A component architecture for high-performance applications. *International Journal of High-Performance Computing Applications*, (14):347–358, 2000.
36. Peter J. Keleher, Jeffrey K. Hollingsworth, and Dejan Perkovic. Exploiting application alternatives. In *19th International Conference on Distributed Computing Systems*, 1999.
37. C. T. Kelley and D. E. Keyes. Convergence analysis of pseudo-transient continuation. *SIAM Journal on Numerical Analysis*, 35:508–523, 1998.
38. Joseph P. Kenny, Steven J. Benson, Yuri Alexeev, Jason Sarich, Curtis L. Janssen, Lois Curfman McInnes, Manojkumar Krishnan, Jarek Nieplocha, Elizabeth Jurrus, Carl Fahlstrom, and Theresa L. Windus. Component-based integration of chemistry and optimization software. *Journal of Computational Chemistry*, 24(14):1717–1725, 15 November 2004.
39. Benjamin C. Lee, Richard Vuduc, James Demmel, and Katherine Yelick. Performance models for evaluation and automatic tuning of symmetric sparse matrix-vector multiply. In *Proceedings of the International Conference on Parallel Processing*, Montreal, Quebec, Canada, August 2004.
40. H. Liu and M. Parashar. Enabling self-management of component based high-performance scientific applications. In *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing*. IEEE Computer Society Press, July 2005.
41. A. Malony, S. Shende, N. Trebon, J. Ray, R. Armstrong, C. Rasmussen, and M. Sottile. Performance technology for parallel and distributed component software. *Concurrency and Computation: Practice and Experience*, 17:117–141, Feb–Apr 2005.
42. D. J. Mavriplis. Multigrid approaches to non-linear diffusion problems on unstructured meshes. *Numerical Linear Algebra with Applications*, 8:499–512, 2001.
43. Michael O. McCracken, Allan Snavely, and Allen Malony. Performance modeling for dynamic algorithm selection. In *Proceedings of the International Conference on Computational Science (ICCS'03), LNCS*, volume 2660, pages 749–758, Berlin, 2003. Springer.
44. L. C. McInnes, B. Norris, S. Bhowmick, and P. Raghavan. Adaptive sparse linear solvers for implicit CFD using Newton-Krylov algorithms. In *Proceedings of the Second MIT Conference on Computational Fluid and Solid Mechanics, Cambridge, MA*, volume 2, pages 1024–1028. Elsevier, 2003.
45. Lois Curfman McInnes, Jaideep Ray, Rob Armstrong, Tamara L. Dahlgren, Allen Malony, Boyana Norris, Sameer Shende, Joseph P. Kenny, and Johan Steensland. Computational quality of service for scientific CCA applications: Composition, substitution, and reconfiguration. Preprint ANL/MCS-P1326-0206, Argonne National Laboratory, Feb 2006. Available via `ftp://info.mcs.anl.gov/pub/tech_reports/reports/P1326.pdf`.
46. D. Mihalas and B. Weibel-Mihalas. *Foundations of Radiation Hydrodynamics*. Dover Publications, Inc., Mineola, NY, 1999.

47. J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag, 1999.
48. B. Norris, L. McInnes, and I. Veljkovic. Computational quality of service in parallel CFD. In *Proceedings of the 17th International Conference on Parallel CFD*, 2005. Also available as Argonne National Laboratory preprint ANL/MCS-P1283-0805 via `ftp://info.mcs.anl.gov/pub/tech_reports/reports/P1283.pdf`.
49. B. Norris, J. Ray, R. Armstrong, L. C. McInnes, D. E. Bernholdt, W. R. Elwasif, A. D. Malony, and S. Shende. Computational quality of service for scientific components. In *Proceedings of the International Symposium on Component-Based Software Engineering (CBSE7), Edinburgh, Scotland*, 2004. Also available as Argonne National Laboratory preprint ANL/MCS-P1131-0204 via `ftp://info.mcs.anl.gov/pub/tech_reports/reports/P1131.pdf`.
50. R. Raje, B. Bryant, A. Olson, M. Augoston, and C. Burt. A quality-of-service-based framework for creating distributed heterogeneous software components. *Concurrency Comput: Pract. Exper.*, (14):1009–1034, 2002.
51. J. Ray, N. Trebon, S. Shende, R. C. Armstrong, and A. Malony. Performance measurement and modeling of component applications in a high performance computing environment : A case study. In *Proceedings of the 18th International Parallel and Distributed Computing Symposium*, April 2003.
52. David Reiner and Tad Pinkerton. A method for adaptive performance improvement of operating systems. In *Proceedings of the 1981 ACM SIGMETRICS Conference on Measurement and Methodology of Computer Systems*, pages 2–10, September 1981.
53. Self-Adapting Large-scale Solver Architecture, see `http://icl.cs.utk.edu/salsa`, 2006.
54. S. Shende and A. D. Malony. The TAU parallel performance system. *International Journal of High Performance Computing Applications*, 20(2):287–331, 2006.
55. Shweta Sinha and Manish Parashar. System sensitive runtime management of adaptive applications. In *Proceedings of the Tenth IEEE Heterogeneous Computing Workshop*, San Francisco, CA, 2001.
56. B. Smith et al. TOPS Solver Components. `http://www.mcs.anl.gov/scidac-tops/solver-components/tops.html`, 2006.
57. M. Sosonkina. Runtime adaptation of an iterative linear system solution to distributed environments. In *Applied Parallel Computing, PARA'2000*, volume 1947 of *Lecture Notes in Computer Science*, pages 132–140, Berlin, 2001. Springer-Verlag.
58. Johan Steensland and Jaideep Ray. A partitioner-centric model for SAMR partitioning trade-off optimization: Part II. In *Proceedings of the 6th International Workshop on High Performance Scientific and Engineering Computing (HPSEC-04)*, August 2004. Held in conjunction with The 2004 International Conference On Parallel processing (ICPP-04), in Montreal, Canada.
59. Johan Steensland and Jaideep Ray. A partitioner-centric model for SAMR partitioning trade-off optimization: Part I. *International Journal of High Performance Computing Applications*, 19:1–14, 2005.
60. Cristian Tapus, I-Hsin Chung, and Jeffrey K. Hollingsworth. Active Harmony: Towards automated performance tuning. In *Proceedings of SC02*, 2002.
61. N. Trebon, A. Morris, J. Ray, S. Shende, and A. Malony. Performance modeling of component assemblies with TAU. Presented at Compframe 2005 workshop, Atlanta, June 2005.
62. Jeffrey S. Vetter and Patrick H. Worley. Asserting performance expectations. In *Proceedings of SC02*, 2002.

63. Richard Vuduc, James Demmel, and Jeff Bilmes. Statistical models for empirical search-based performance tuning. *International Journal of High Performance Computing Applications*, 18(1):65–94, February 2004.
64. Richard Vuduc, James W. Demmel, and Katherine A. Yelick. OSKI: A library of automatically tuned sparse matrix kernels. In *Proceedings of SciDAC 2005*, Journal of Physics: Conference Series, San Francisco, CA, June 2005. Institute of Physics Publishing.
65. R. Clint Whaley and Antoine Petitet. Minimizing development and maintenance costs in supporting persistently optimized BLAS. *Software: Practice and Experience*, 35(2):101–121, February 2005. `http://www.cs.utsa.edu/~whaley/papers/spercw04.ps`.
66. R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra. Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001. Also available as University of Tennessee LAPACK Working Note #147, UT-CS-00-448, 2000 (`www.netlib.org/lapack/lawns/lawn147.ps`).
67. K. Whisnant, Z. Kalbarczyk, and R. K. Iyer. A foundation for adaptive fault tolerance in software. In *Proceedings of the 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, pages 252–260, April 2003.
68. Eric Wohlstadter, Stefan Tai, Thomas Mikalsen, Isabelle Rouvellou, and Premkumar Devanbu. GlueQoS: Middleware to sweeten quality-of-service policy interactions. In *Proceedings of the 26th International Conference on Software Engineering (ICSE '04)*, pages 189–199, May 2004.
69. K. Zhang, K. Damevski, V. Venkatachalapathy, and S. Parker. SCIRun2: A CCA framework for high performance computing. In *Proceedings of the 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2004)*, Santa Fe, NM, April 2004. IEEE Press.